

**REMARKS**

We have carefully considered the Office Action dated May 2, 2006, in which all claims are rejected and the Advisory Action dated August 15, 2006. In the Response to Amendment section of the Office Action the Examiner states that “it is noted that the features upon which the Applicant relies are not recited in the rejected claim.” As an example the Examiner states “Applicant’s contend ... ‘Fuoco does not teach or suggest generating and using parity check bits that are stored in a *separate* buffer location’ ...” (emphasis in original). While we contend that the claims pending prior to this Amendment stated that the data and parity bits were stored in separate buffer locations, we have amended the independent claims to further clarify this point. Accordingly, there can be no doubt that the data words are stored in data buffer locations and the parity check bits are stored in parity check locations, which are different locations than the data buffer locations that contain the data. Specifically, claim 1 now reads in part

storing the data words in a plurality of **data buffer locations** in the buffer memory and the parity check bits in one or more **parity check buffer locations** in the buffer memory, **the parity check buffer locations being different locations than the data buffer locations that contain the data;**(emphasis added)

which clearly recites this relied-upon feature that is not taught by Fuoco. Claim 1 also includes the step of

producing from the stored and regenerated parity bits a result that is usable **to directly identify the data buffer location that contains of a data code word that contains an erroneous bit** and the position of the erroneous bit in the data word contained in the identified data buffer location. (emphasis added).

which is another clearly recited relied-upon feature that is not taught by Fuoco. As set forth in the prior response, there is no teaching or suggestion in Fuoco of producing such a result. Indeed, there is no teaching or suggestion in Fuoco of the claim 1 step of

applying data to be stored in a buffer memory **as a plurality of data words** to a generator matrix to generate parity check bits; (emphasis added).

Thus, the current invention encodes many data code words to produce one set of parity bits. In contrast, Fuoco separately generates the check bits that correspond to respective 4-byte data code words. See, Column 5, lines 29-54 and Table 1. As stated in Fuoco “the ECC logic will also calculate 7 check bits for each data code word.” (emphasis added) Column 4, lines 5-6. See also Table 1 and Column 5, lines 29-54.

In particular, the Fuoco reference describes a system in which the respective memory locations are 40 bits wide, and “each memory location ... stores a 4 byte data word, the associated 7 check bits and the flag bit for each data word.” Column 4, lines 6-9. See also, Column 3, lines 24-33; Column 5, lines 58-60. Thus, there can be no doubt that the Fuoco reference teaches storing the data and associated check bits in the *same memory location*, and that independent claim 1 quoted above as well as the remaining independent claims respectively recite storing the parity check bits **in parity check buffer locations that are different locations than the data buffer locations in which the data are stored**.

The RAS, CAS WE, OE and ADDR signals to which the Examiner refers are discussed in Fuoco as being used for a conventional write operation in which an entire 4 byte data word is overwritten and also for a read-modify-write operation that involves

overwriting only certain of the four bytes of a data word. The 7 check bits are generated for the data word in either write operation and the check bits are stored then in the **same memory location** as the particular data word to which the check bits correspond. See, Column 4, lines 44-46 and 56-61; Column 8, lines 20 et seq.

In the read-modify-write operation, the row and column activation strobes, RAS and CAS respectively are used to properly produce the new data word and then encode the data code word, to produce the associated 7 check bits that will ultimately be stored in the memory location with the 4 byte or 32 bit data word and the associated system flag, to fill the 40 bit wide memory location.

As described beginning at Column 7, line 60, when the entire data word stored in a given memory location is not to be overwritten, the data word is first read from that memory location and corrected as necessary using the check bits read from the same memory location. The corrected data word is then supplied to the data latch 52.

The data word supplied to the data latch 52 and the data bytes that are to be newly written, that is, the bytes that are to overwrite certain bytes of the retrieved data word, are supplied to the multiplexer 40, which is activated by the CAS signal. The multiplexer multiplexes the new data bytes with the data bytes stored in the data latch, to form the new four byte data word that is to be written to the memory location.

The new four byte data word is provided through the selector 42 to the check bit generator 44, which generates the new seven check bits, which are then written to the **same memory location** that contains the new data word. See, Column 8, lines 15-20.

Thus, Fuoco clearly states that the data word and the associated check bits are stored in the *same memory location*, whether the check bits are generated during a conventional write operation or during a read-modify-write operation. See generally, Column 4, lines 56 - Column 5, lines 57; also Column 5, lines 58-59.

There is thus no teaching or suggestion in Fuoco that the check bits are stored in **“one or more parity check buffer locations in the buffer memory, the parity check buffer locations being different locations than the data buffer locations that contain the data,”** as is stated in claim 1 (emphasis added).

Further, the Fuoco system clearly does not perform the step of producing from the stored and regenerated parity or check bits a result that is “usable to directly identify the data buffer location of a data word that contains an erroneous bit” (quoting claim 1). In contrast, the Fuoco system uses check bits that are generated from and correspond to a single data word. Further, only a single data word is manipulated to regenerate the check bits that are used along with the stored check bits in the error detection and correction operations. As discussed, the data word and the stored check bits are stored in and read from the *same memory location*. See, e.g., Column 5, lines 58 et seq. Thus, there is no need in Fuoco to specify through the error detection and correction operations the data buffer location in which the data word that contains an error is stored - since the Fuoco system by definition knows the location of the data code word that is manipulated to produce the parity check bits.

As discussed above, the invention of claim 1 operates in a completely different manner. The inventive method generates the parity bits by encoding all together “a

*plurality of data words*” (quoting from claim 1). The data words are then stored “in a plurality of data buffer locations in the buffer memory” (quoting claim 1). The parity bits, which correspond to the plurality of data words are stored “in one or more parity check buffer locations in the buffer memory, the parity check buffer locations being different locations than the data buffer locations that contain the data.” (quoting claim 1 emphasis added) The stored data, i.e., the plurality of data words, and the parity check bits are read from their respective data and parity check memory locations, and the plurality of data words are again encoded all together to regenerate the parity bits. The stored and regenerated parity bits, which apply to the “plurality of data words” (see first paragraph of the claim 1), are then manipulated to produce the result that is “useable to directly identify the data buffer location of a data word that contains an erroneous bit” (quoting claim 1) i.e., the location that contains a particular one of the plurality of data words that comprises the data of claim 1.

The Examiner in rejecting claim 3 and (cancelled) claim 4 states that the Fuoco “substantially teaches ... the address locations in the add-on memory are assumed to be 40 bits wide and the data words are written as 4 bit strings with 7 check bits generated thus accounting for the 39 of the possible 40 bits in each address” (Office Action page 6). Accordingly, the Examiner **agrees** that each data word and its associated check bits are stored in the same 40-bit wide memory location in the Fuoco system. This is in total contrast to claim 1, from which claim 3 depends. While the quotation from Fuoco does not pertain to the limitation added by claim 3, we again point out that the Fuoco system has no need to produce, as a result of the comparison of the stored and regenerated check

bits, information that identifies the particular data buffer location that contains a data word with one or more erroneous bits. In the Fuoco system, the data word and the associated check bits that are used during the error detection and correction are read from the *same memory location* and thus, the address of the data word is by definition known.

Claims 5 and 6 add a limitation relating to the generator matrix that is used to encode the plurality of data words. As specifically stated in claim 5, the parity check generation portion of the generator matrix **“comprises rows of bits corresponding to binary representations of the data buffer locations used to store the data.”** There is no teaching or suggestion of this in Fuoco. Rather, Fuoco describes the generation of the check bits that correspond to a single data word using a table that illustrates how the bits of the data word participate in the encoding. See, Table 1 in Column 5 and 6 of Fuoco. As can be seen from Table 1, there is no teaching or suggestion of a generator matrix in which the parity check generation portion “comprises rows corresponding to binary representations of the data buffer locations used to store the data” (quoting claim 5). Indeed, if such a matrix were used in Fuoco, the matrix would have to change for each data word - since each data word is **separately encoded**.

We do not specifically address the Examiner’s rejections of the remaining claims that depend from claim 1. This should not be construed as acquiescence to the rejections, but as recognition that the rejections are moot based on our remarks regarding the allowability of the independent claims and the discussion of dependent claims 2-6. However, we point out with respect to claim 12, that the Examiner has correctly stated, in his rejection of claim 4, that the Fuoco system “corrects single bit errors.” See, also,

Column 5, lines 38-39. The syndrome decode logic of the Fuoco system also detects multiple bit errors but cannot correct them. See, Column 6, lines 29-33. Accordingly, the Fuoco system does not teach **“identifying the positions of two erroneous bits of the data code word,”** as is explicitly stated in claim 12, and which necessarily results in the correction of the two bits by flipping the bits.

The discussion above applies also to the remaining method and apparatus claims. Specifically, there is no teaching or suggestion in the Fuoco patent of a method that indicates the step of: **“applying data ... to a generator matrix as a *plurality* of data words to generate parity check bits.”** (quoting claim 23 emphasis added). Further there is no teaching in Fuoco of a generator matrix **“comprising a data portion and a parity check generation portion, and the parity check generation portion *comprises rows of bits corresponding to binary representations of the respective data buffer locations to be used to store the data words.*”** (quoting claim 23 emphasis added). In addition, there is no teaching in Fuoco of **“storing the parity bits in one or more parity check locations of the buffer memory, the parity check buffer locations being different locations than the data buffer locations that contain the data.”** (quoting claim 23 emphasis added). Rather, as discussed above and in the Office Action, Fuoco describes a memory in which a four-byte data word and the 7 check bits generated by the encoding of the data word are stored in the *same memory location*. See, Column 3, lines 28-34; Column 4, lines 6-9; Column 5, lines 58-60, Office Action, page 6. Thus, there can be no question that the encoding method of claim 23 is not taught or suggested by the Fuoco patent. Further, the encoding in Fuoco does not produce a single set of parity

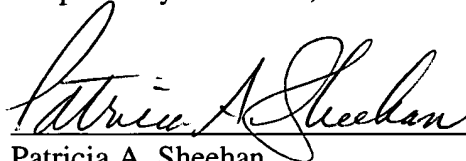
check bits from the encoding of a plurality of data words, as set forth explicitly in independent claim 23.

Further, the data storage system and the apparatus of independent claims 24 and 26 include controllers that are operable to perform the steps of claim 1, and thus, the discussion as to the lack of teaching by Fuoco of particular steps of claim 1 applies equally to these claims. Similarly, the data storage system and the apparatus of claims 25 and 27 includes a controllers that are operable to perform the steps in claim 23 and the discussion of the lack of teaching by Fuoco as to particular steps of claim 23 applies equally to these claims.

We believe that the discussion set forth above, which relies specifically on the language of the pending claims, shows that the invention as set forth in the claims is not taught or suggested by Fuoco. In light of the above, we ask that the Examiner reconsider the rejections and issue a Notice of Allowance for all pending claims, as amended to correct an omitted dependency in claim 21.

Please charge any fee occasioned by this paper to our Deposit Account  
No. 03-1237.

Respectfully submitted,



Patricia A. Sheehan  
Reg. No. 32,301  
CESARI AND MCKENNA, LLP  
88 Black Falcon Avenue  
Boston, MA 02210-2414  
(617) 951-2500